University of Amsterdam

MSc System and Network Engineering

Cyber Crime and Forensics

Forensic analysis of the Nintendo Wii U

Authors:

Romke van Dijk
romke.vandijk@os3.nl

Dana Geist
Dana.Geist@os3.nl

March 30, 2016

Abstract

This report shows a forensic analysis of the Nintendo Wii U console, which aims to determine what data can be extracted to serve as forensic evidence. To conduct this research we focused on post mortem and live forensics. We showed that in order to perform proper post mortem forensics, a chip off procedure is required. In addition, we performed a live forensics analysis by means of a user level exploit. Throughout the research we showed how to extract memory and files. From the memory analysis, we could find traces of recently visited web sites and file system paths. Using those paths, we were able to access a file containing browsing history which is not accessible from the user interface. We also showed the Wii U does not implement proper user separation from the aforementioned browser history. This implies that even if an account is password protected, its browsing history can be accessed from another account. Moreover, we showed the forensic relevance of each of the Wii U features. Finally, we proposed a procedure that can be used as a starting point for a forensic analysis on the console.

Contents

1. Introduction

1.1 Motivation

The current generation of game consoles are designed to be entertainment systems but they offer more than just gaming capabilities, and are as powerful as personal computers. They offer functions such as interacting with other gamers, browsing the web, or watching movies. The social interaction and the money involved has led criminals to seek victims in the online gaming world. West [Wes15] showed in his research that the criminal activities can vary from money laundry and extortion, to kidnapping, molestation and murder. This makes gaming consoles relevant for forensic investigators.

The Nintendo Wii U is a gaming console that provides online features such as instant messaging and video chatting, which allow for different types of inter-user communication. As the Wii U is a console better suited for kids (because of the extensive parental control it provides), it is a real threat that criminals can use it to contact them. In this context, it becomes relevant to study this gaming console from a forensics perspective, in order determine what type of data can be extracted from it that can be of value in a forensic investigation.

As it can be seen in the "Related Work" section, there is no published forensic research on the Wii U console. Therefore, by conducting this research, we intend to fill this knowledge gap.

1.2 Research Questions

Our main research question is: *What data can be gathered from the Nintendo Wii U console to serve as forensic evidence?*.

In order to aid this forensic research we subdivided the main question into two sub-questions:

- What type of information can be obtained using post mortem analysis?
- What type of information can be obtained using live analysis?

1.3 Related Work

As gaming consoles became relevant for forensic investigations, there has been research done on many of them in the past. Burke and Craiger [BC07] performed a forensic analysis on the Xbox. Moore et al. [Moo+14] built on top of this research and analysed the Xbox One. Recently, a first initial

research was performed on the Playstation 4 by Davies et al. [Dav+15]. For Nintendo's platforms, only the Wii was researched by Turnbull [Tur08] and Stewart [Ste10].

The Wii U also got the attention of the hacking group Fail0verflow who allegedly managed to run arbitrary code on the console by means of a kernel level exploit [Fai14]. Their work gave insights into the inner workings of the Wii U and provided important basis for further research. Another relevant resource has been the Wii U Brew Wiki [Wii16f]. It provided detailed information about the memory structure, system libraries, function calls and other internal operations. However, those mentioned sources are not focused on forensic analysis of the Wii U, but more aimed at running arbitrary code on the console.

1.4 Scope

This research consists of performing a forensic analysis on the Wii U console, firmware version 5.5.0, released on the 17th of August 2015 [Nin16a]. Throughout this investigation, we focused on analysing different aspects of the Wii U such as memory, file system and user interface. The objective is to be able to determine what information can be extracted from the Wii U, that can serve as forensic evidence.

1.5 Structure

This paper will first start by describing the methodology used, in chapter 2. Chapter 3 will focus on the post mortem analysis, while chapter 4 will focus on the live analysis and will describe what data can be gathered from the features offered by the Nintendo Wii U. Chapter 5 will summarize the main findings. All the information found throughout the research will be combined to provide a procedure that can be used a starting point for a forensic analysis of the console. This procedure is described in chapter 6. Chapters 7, 8 and 9 will talk about the conclusion, discussion points and future work respectively.

2. Methodology

In order to investigate what type of information can be obtained from the Wii U, we focused on post mortem and live analysis. For post mortem forensics, we looked at the architecture of the console. The main objective of this phase was to study which physical components could store relevant information, and how this information could be extracted (if possible). For this, we looked at different sources such as FailOverflow [Fai14] and Wii U Brew [Wii16f]. The mentioned sources helped us identify interesting components, possible roadblocks and gave us ideas on how to approach the research.

As the Wii U is a proprietary and closed system, there is no straightforward way of performing live forensics on it. Nintendo tried to protect the console from piracy, making it difficult for users to run unauthorized code on it. This is the reason why, for this phase of the investigation we decided to use a publicly available user space exploit [Wii16g]. By this means, we were able to run our own code and get partial access to different data sources such as the memory and different files stored on disk. In addition, we identified several graphical user interface features the Wii U offers, that could become relevant in a forensic investigation. The idea was to explore what kind of information could be extracted by simply walking through the user interface. Examples of these features are: web browser, Friendlist, Miiverse messaging, among others.

Finally, we provided a procedure that can be used as a starting point for a forensic analysis on the console.

3. Post Mortem Forensics

Post Mortem forensics refers to the forensic methods used on powered off devices. Analysing hard drives and massive storage devices is part of this type of investigation. Even memory can be studied using a post mortem approach if forensic investigators act fast enough. However, this is out of the scope of this research.

The Wii U has multiple ways of storing data on non volatile devices. It has two 512MB NAND flash storage components. One of them is used to store part of the Wii U operating system and the other one is used for retro compatibility functions with the Wii. Besides these NAND flash components, it has either a 8GB or a 32GB eMMC storage component depending on the console version [Fai14]. In addition, it offers several USB ports to attach USB mass storage devices, and it is also possible to use a SDCard for some specific functions. Figure 3.1 shows the internals of the Wii U.

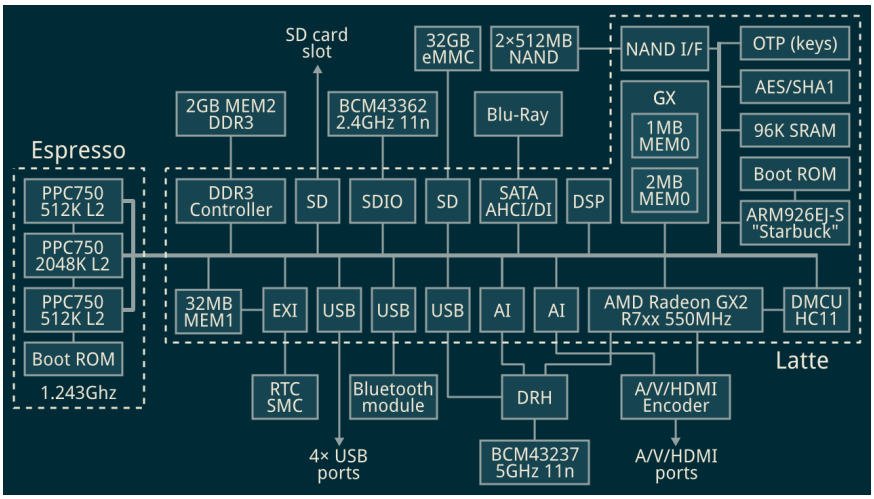

Figure 3.1: Block diagram of the internals of the Wii U [Fai14]

From inside the user interface there is no direct way of accessing the filesystem, except from the 'data management' option. This option allows for moving data between USB devices and the internal eMMC component. Data comprises games, save files containing information related to played games and applications. Analysing the eMMC and NAND components further would require performing a chip off procedure, as they are not tra-

ditional hard disk drives. This procedure is expensive and needs specific tooling. Therefore, it was considered out of the scope of this research.

3.1 External Storage Devices

As explained before, it is possible to move data from the internal storage to a USB mass storage device. To investigate this, the following experiment was performed.

- A wiped Wii U with firmware version 5.5.0E and with one user was used.
- Mario Kart 8 was launched and was played for roughly 15 minutes to create a save game.
- A wiped 8GB USB mass storage device was attached to the Wii U and formatted by the Wii U.
- The save game was moved from the eMMC to the USB mass storage device through the data management option.
- The USB was inserted into a forensics ready computer, running Deft 8.1 and an image of it was created using the `dd` command.

First of all, we tried mounting the USB image using the Linux `mount` command but no filesytem was recognized. Therefore, we decided to analyse it with a hexadecimal editor, to visualize the raw data. We found most of the sectors to be empty. However, some sectors did contain data. Table 3.1 shows the layout of these sectors. The entropy of the data inside those sectors was analysed using the `ent` command [Wal08]. Its output showed that the information density was high. No evidence was found of which file system was used. Either the content of the USB was encrypted or highly compressed. The research conducted by Fail0verflow shows that AES encryption is being used by Nintendo for several functions inside the Wii U, which could imply that this is also the case for the USB. Besides from the entropy, `ent` gave more indicators showing that the data is encrypted, such as the Chi square distribution and the Monte Carlo value of Pi. Both values are also within ranges of good pseudo random number generators. Listing 3.1 shows the output of the `ent` program when it was run on the first part of the USB.

```
Entropy = 7.997955 bits per byte.

Optimum compression would reduce the size
of this 94207 byte file by 0 percent.

Chi square distribution for 94207 samples is 267.37,
and randomly would exceed this value 50.00 percent
of the times.

Arithmetic mean value of data bytes is 127.3507 (127.5
= random). Monte Carlo value for Pi is 3.150117827
(error 0.27 percent). Serial correlation coefficient is
0.006274 (totally uncorrelated = 0.0).
```

Listing 3.1: Ent output of a part of the USB

Start offset (hex)	End offset (hex)	Size	Entropy (bits in bytes)
0	16FFF	94kB	7.997955
20000	23FFF	16kB	7.986486
26000	2C000	25kB	7.993083

Table 3.1: Layout of the USB image after storing a Mario Kart 8 save game

Besides the normal Wii U operating system, the Wii U contains the vWii mode, which is the Wii mode where Wii software can be run on the Wii U hardware. The SDCard is mostly used for the vWii mode and can also be used to store input images for the MiiMaker application. Because our research in focused on the Wii U, everything related to the vWii was considered to be out of scope.

4. Live Forensics

Live forensics is a method that involves gathering data from a system which is still running. It usually allows for accessing more information than post mortem forensics methods as it involves volatile information retrieval. As volatile information changes fast, it is important to prioritize acquisition, to be able to collect as much information as possible.

Even if live analysis is currently a widely accepted forensic method, it has some disadvantages. First of all, data can be modified during the acquisition process. Secondly, the acquisition is performed by programs running in user space by requesting information from the operating system. This means that the acquisition program can be vulnerable to modified operating systems, programmed to implement anti-forensics methods [Jon07].

There are multiple components that are volatile such as registers, cache and memory. In this research the focus is on the memory of the Wii U which will be explained in detail in section 4.2.

After following the procedure for dumping the memory, the forensic investigator needs to continue with the extraction of information from the file system as per section 4.3. Even if this information is not stored in a volatile piece of hardware, it can be changed by programs running on the console. Therefore, it is important to retrieve this information as soon as possible, to avoid further modification of the evidence.

Finally, we describe what valuable information is available through the graphical user interface, which is detailed in section 4.4. This task is not so critical time wise as the previously mentioned tasks, simply because the information is not likely to change that fast if the console is properly isolated. However, some features do require an internet connection to the services, because the data of those applications is hosted by Nintendo.

4.1 Accessing the system

4.1.1 Exploit

Nintendo Wii U is a closed proprietary system. Therefore, its structure and inner workings are not well documented. In this context, it is very difficult to be able to access and explore the system as there is little knowledge about its architecture. However, there are active communities interested in the Nintendo Wii U, which try to shed some light on the system's hardware and software, such as FailOverflow and Wii U Brew. Unfortunately, most of them are aimed at running illegal software, so they are not focused on providing forensics tools.

In order to access and interact with the system, we needed to run code on it. FailOverflow claims to have developed a kernel level exploit which would give access to most system components, but they did not release it to the

general public. As this was not an option, we found a publicly available open source user level exploit for the latest firmware versions (5.5.0E and 5.5.1E), which gave us the ability to run our own code in the console. By combining the framework created by Libwiiu [Wii16g] and the scripts to generate the payload created by Yellows8 [Yel16] we were able to implement this so-called *Wiiuhaxx* exploit. It is a return oriented programming chaining exploit [Wik16a] which allows for running binary code in the user space. It is based on a vulnerability present in the mp4 player, which is offered by the web browser.

At this point, it is important to note that the code was developed in the C language and cross compiled for PowerPC which is the specific architecture the Wii U processor has [IGN16]. In addition, the development of code was limited to the native system libraries that the Wii U offers. This basically means that simple C functions such as `printf` are not available, as it is not possible to link the shared libraries required to execute them.

The Wii U's loader is the component responsible for loading the native libraries and executables into memory. All libraries are RPL files, which constitute a modification of the standard ELF format with compressed sections. Once loaded, applications dynamically link to them to get access to different operating system services. Services included are memory management, graphics, audio and controller input [Wii16a].

There are many RPL libraries provided by the Wii U. However, most of them are not documented, so it is not possible to know which library functions they offer. A list of the different known RPLs can be found in [Wii16a].

4.1.2 Set Up

In order to implement the exploit, we set up a web server which served the payload containing the binary C code we wanted to execute on the console. Once the server was in place, we were able to access the payload from the Wii U by using the web browser and accessing the payload url from it. To be able to access the web server from the Wii U, we connected it to a router via WiFi. The web server, in turn, had a wired connection to the router. Figure 4.1 illustrates this set up.

Figure 4.1: Set Up

4.1.3 RPC

The next step in the process was to be able to communicate with the running program in the Wii U, so as to manage it interactively. The solu-

tion to this, was to use the RPC (remote procedure call) framework built by Libwiiu [Wii16g] and ported to the newest firmware by NWPlayer123 [NWP16]. This framework consists of an interactive python program running on the web server, which waits for the C program running on the Wii U to connect to a specific socket. Once this tcp connection is established, python functions can be used to trigger actions on the C program.

The system library which enables the implementation of the tcp connection logic inside the Wii U, is the `Nsysnet.rpl` library [Wii16c].

Some of the functions provided by this library are connect, send and receive as shown below:

```
/* Socket functions */
int (*socket)(int family, int type, int proto);
int (*connect)(int fd, struct sockaddr *addr, int addrlen);
int (*recv)(int fd, void *buffer, int len, int flags);
int (*send)(int fd, const void *buffer, int len, int flags);
```

In the python side, this connection was implemented simply by using the socket standard library.

4.1.4 Debugging

Running our own C code on the Wii U console was a difficult task because the environment did no offer any debugging possibilities. As this was crucial to understanding issues, we decided to leverage the Wii U screen, to render debugging messages. We managed to implement the text rendering on the main screen and the Gamepad by using the `coreinit.rpl` system library [Wii16b]. Some of the functions we used are:

```
/* OSScreen functions */
void(*OSScreenInit)();
unsigned int(*OSScreenGetBufferSizeEx)(unsigned int bufferNum);
unsigned int(*OSScreenSetBufferEx)(unsigned int bufferNum,
void * addr);
```

4.2 Memory

As shown in Figure 3.1, the Nintendo Wii U has 2GB of DDR3 RAM memory consisting of four 512MB DRAM chips [Wik16b]. From these 2GB of RAM, 1GB is allocated to system functions, while the rest is reserved for games [IGN16]. The system functions include running applications such as Miiverse and the web browser. This part of the memory is the most interesting one for this research as these applications can be misused.

Even though the Wiiuhaxx exploit gave us access to user space memory, we needed to find a way of exfiltrating data from the system. There were limited possibilities to do so, as the SDcard is mostly used for the Wii mode and the USB is most likely encrypted by the Wii U, as explained before. Another possibility would be printing the memory on the screen and then

take pictures of it. However, this is not a very practical solution. Therefore, we decided to use the RPC framework to transfer the memory data to our web server. The RPC implementation we used, already provided different functions to read the memory, such as `dump_mem`.

```
def dump_mem(self, address, length, filename):
```

This function expects as input parameters the `address`, which is the memory address from which to start reading, and `length` which specifies how many bytes should be read. As an output parameter the function returns a file called `filename`, with the requested memory contents.

It is important to note that this function uses the socket standard library to send the requests and receive the data. This means that the C program, in turn, also needs to be programmed to use the socket functions provided by the `nsysnet` RPL.

Before extracting the memory, we needed to determine the value of the parameters expected by the function showed above. We did so, by combining our knowledge about how the Wii U memory is handled, with the virtual memory map provided by Wii U Brew [Wii16a]. On one side, we were able to find that the virtual address for the application and library area starts from the initial value of 0x10000000, so we had the lower limit. On the other side, we knew that the memory allocated for running applications was 1GB, so we were able to determine the upper limit (amount of bytes) as well.

Putting all these together, we extracted the memory data by executing the `dump_mem` function with the correct parameters.

```
# python3 -i rpc.py
Listening...
>>> rpc.dump_mem(0x10000000, 1000000000, 'all.bin')
```

Listing 4.1: Python function for memory dump

4.3 File System

After analysing the memory, the next step was to analyse the file system. As described in section 3, the graphical user interface offers limited access to it. Therefore, we used the setup described in section 4.1, to run code and be able to access directories and files. Documentation showed that the Wii U has system libraries available to access the file system [Wii16a], so we used those. Moreover, the Libwiiu project [Wii16g] contains sample code implementing several file system operations. However, the examples did not work on the latest firmware. This required us to work on the code and port it to run in the newest firmware, so we could implement our own operations on top of it. Below two example file system operations are shown.

```
/* FS functions */
int (*FSOpenDir)(void *client, void *cmd, char *path, u32
*dir_handle, int unk1);
```

```
int (*FSReadDir)(void *client, void *cmd, u32 dir_handle,
void *buffer, int unk1);
```

As described in section 4.1, the RPC script was used to interact with the Wii U. This RPC script was extended to support the following operations:

- `ls`: ability to list the contents of the directory.
- `cat`: ability to copy a file over the TCP connection to the remote server.
- `scp`: ability to copy files from a specific directory over the TCP connection to the remote server.
- `dumpfs`: takes a list of directories as input and copies all its content to the remote server.

The directory structure used in the Wii U has a specific scheme. Each application has a unique ID which consists of 2 parts of 4 bytes each. As an example, the web browser has the following ID: `00050030-1001220A`. The first part indicates the general application group and the second part is the specific ID for the application itself. The IDs differ depending on different releases of the Wii U (there are different versions for Japan, USA and Europe) [Wii16e].

Most of the data is stored on the eMMC component, which seems to be mounted in the file system as `/vol/storage_mlc01/`. Our analysis showed that inside this directory there are two sub directories: `sys` and `usr`. The `usr` directory stores the user save files of an application, but only if the application requires this. For some applications the save file is not stored in the `usr` folder. Instead, a 4 byte unique ID is generated per user and the save data is stored in `/save/common/<unique id>/`. The `sys` folder contains application data and the specific ID of an application is used to refer to it. For example, data related to the web browser is stored in the `/vol/storage_mlc01/sys/title/00050030/1001220A/content` directory. When a specific application is run, the directory with all the application data (normally stored in the specific `sys` folder) is mounted or symlink at `/vol/content`.

As the exploit we used only granted access to the user space, we expected to encounter some specific files and directories which we would not have access to. We were able to confirm this by checking the error codes returned by the Wii U, while executing the `ls` command. This way, we could distinguish between non-existing and restricted files and directories.

4.4 Graphical User Interface

As described in section 1.1, the Wii U offers more online features than any Nintendo console has ever had. The common factor in most of the services is the Nintendo Network ID, which is used to connect them. The

Nintendo Network ID also allows for a password to be configured to get access to that specific user's session in the console. This involves accessing the user's personal data. In this section we describe the functionality of each of the features, and what data might be interesting for a forensic investigator to look into.

4.4.1 Web Browser

The Wii U web browser implements the NetFront browser engine [Nin16b], which is a popular browser engine for mobile devices. The browser can be used for visiting websites, but it also allows for users to save bookmarks and login credentials. The browser does not show any browsing history, but it shows the most used bookmarks. Open tabs are saved when the browser (or console) is shutdown and those tabs are reopened when the browser is launched again.

4.4.2 Wii U Chat

The Wii U Chat offers video chatting services between Wii U's users. It lists the people who are in the friend list of the current logged in Nintendo Network ID. It also shows the amount of calls made to a specific friend as well as the list of missed calls.

4.4.3 Mii Maker

The Mii Maker offers limited forensic value from our point of view, as it simply allows for a user to create and select a digital avatar, also referred to as Mii's). However, in the UK there has been an incident in which a man discovered his wife was being unfaithful through the addition of Mii character [Tur08].

To create a Mii, a user can take a picture of himself and it will automatically be converted in a cartoon version of the picture. This avatar can then be linked to a Wii U account, either to a local one or to a Nintendo Network ID. The Mii's are used to refer to the account in almost all applications. Mii's can also be transferred via a QR-code between systems.

4.4.4 eShop

The eShop is the digital Nintendo store were users can download demo's, watch trailers, buy games and more. It might show some digits of the credit card used to buy certain content. This information could be potentially used to relate the credit card with other purchases. In addition, it does show which games have been bought for a specific Wii U. It is important to note that the bought content is not user specific, but is Wii U specific.

4.4.5 Miiverse

The Miiverse can be considered the heart of the social interaction on the Wii U. It can be compared to an online forum. Users can join multiple

communities (which are game specific) and post either messages or replies. Those messages can be either text or drawings. Users can also be followed or follow other users, create and save screenshots, create game journal entries to show progress in a specific game or send private message to friends. This feature could potentially show relations between people. To use this feature, a user has to be able to connect to the Nintendo Network. Depending on the privacy settings of the user, some features might not be available. In addition, it is important to note that the Miiverse can be accessed trough a web browser, so data can be viewed and modified from this interface as well.

4.4.6 Daily Records

The Daily Records is a feature that keeps track of how much a specific game or application has been used by a user. It shows the date the application was used and for how long, but it does not include timestamps. This means, for example, that from a game usage of 4 hours it is not possible to distinguish whether the user was continuously playing for that amount of time, or if the user played for 2 hours on two different points in time.

4.4.7 Friendlist

The Friendlist shows the friends of a user, who of those are online and what game they are currently playing (this information may not be available depending on the user privacy settings). Users can also set status messages. In addition, the Friendlist shows the last 100 friends who were gaming online with the user.

4.5 Experiments

To see what information can be retrieved using live forensics, the experiments below were performed, using two newly created Nintendo Network IDs.

- Web browser: visit specific URLs and access images.
- Friendlist: log in with one Nintendo Network ID and add the other Nintendo Network ID as a friend. Set a status message.
- Miiverse: send messages between the two Nintendo Network IDs and post messages in communities.
- Mii Maker: create a new Mii by taking a picture.
- Daily Records: access this application and check for dates and played games.

After executing those tests, the memory was dumped using the method described in section 4.2. In order to analyse the collected memory images

`strings` and `scalpel` commands were used. The file system was analysed using the method described in section 4.3. The objective was to find any traces of the performed actions.

5. Findings

5.1 USB

As already explained in section 3.1, we found no evidence of what file system was used for data storage on the USB. In addition, we found that the content of the USB was either encrypted or highly compressed. This means that the USB investigation resulted in a dead end. Without the encryption keys, we could not find a way of analysing the stored contents.

5.2 Memory

By analysing the memory dump extracted from the Wii U, we were able to find traces of recently visited websites, file system paths, and system libraries. Listing 5.1 shows the URL used to access the payload and some filesystem paths found in memory.

```
$ strings all.bin | grep http://
...
http://192.168.1.132/payload/wiiuhaxx.php?sysver=550
...


$ strings all.bin | grep -i /vol/
...
/vol/save/80000006/

/vol/storage_mlc01/sys/title/0005001b/10056000/
content/FFLResMiddle.dat

/vol/content/audio.wav
...
```

Listing 5.1: Memory traces

5.3 File System

The file system analysis was mainly based on the file system paths found during the memory analysis phase. We were able to exfiltrate different files and directories by using the `ls`, `cat`, `scp` and `dumpfs` functions implemented. The most interesting file found was `psv.bin`. For our user it was located in the `/vol/save/80000006/` path, but this path is different per user. This file contains several visited URLs, which seem to appear in chronological order. It is important to note that the web browser application does not offer a way of displaying this browsing history (it only shows the most frequently used bookmarks).

```
>>> rpc.ls("/vol/save/8000006")
Psv.bin
>>> rpc.scp("/vol/save/80000006")
>>> rpc.cat("/vol/save/8000006/psv.bin")
```

Listing 5.2: List and exfiltrate psv.bin file

We also found directories containing Mii avatar related files. In addition, we were able to list directories related to the web browser, which have information such as error messages, default pages and a list of digital certificates. Nonetheless, those files only contained application data and no user data. Listing 5.3 shows the Mii avatar related files and listing 5.4 shows some of the web browser related directories.

```
/vol/storage_mlc01/sys/title/0005001b/10056000/content/
    FFLResHigh.dat
/vol/storage_mlc01/sys/title/0005001b/10056000/content/
    FFLResHighLG.dat
/vol/storage_mlc01/sys/title/0005001b/10056000/content/
    FFLResMiddle.dat
```

Listing 5.3: Mii avatar related files

```
/vol/storage_mlc01/sys/title/00050030/1001220A/content/message/
/vol/storage_mlc01/sys/title/00050030/1001220A/content/pages/
/vol/storage_mlc01/sys/title/00050030/1001220A/content/favicon/
```

Listing 5.4: Web Browser Structure

Throughout our research, we encountered that it was possible to access other users' `psv.bin` files, when logging into the Wii U with a different user. This implies that there is no proper application compartmentalization in place, so one user can get access to another users' browsing history even for users that enabled password protection. This vulnerability can be useful for forensic purposes; if an account that needs to be investigated happens to have a password, it is possible to create a new user (or use an existing user without a password), log in with the new user and extract the `psv.bin` file of all the users of the system.

5.4 Graphical User Interface

The graphical user interface can be analysed in order to find links between people, messages being send and received, and even can be used to help creating a profile of a user. The information that can be extracted from the different features of the Wii U, as described in more detail in section 4.4.

6. Procedure

This chapter shows a procedure that can be used to perform a forensic analysis of the Nintendo Wii U. It is partially based on the procedure described by Turnbull [Tur08]. In order to follow this procedure it is required to have the setup described in section 4.1.

1. Activate logging mechanism and recording device.
2. Make sure the Wii U is in an isolated environment (not connected to the internet).
3. If the console is turned off, turn it on.
4. If the start up screen shows multiple users, record the users. Try to access the first user, if this user is password protected try to access the next user until a user that is not password protected has been found.
5. If all users are password protected the following procedure can be used to get access to the console.
   (a) In the user select screen press "Add New User" and follow instructions, continue from step 6, steps 15-22 do not apply. This is because no relevant information is going to be found for the new user, and there is no GUI access for password protected users.
6. Open "System Settings", in the top right corner it shows the firmware version. Make sure it is either 5.5.0E or 5.5.1E. This research has only been tested for those versions.
7. Record the time and date.
8. Record all network information.
9. Connect to the SSID of WiFi router used in the forensic setup. The Wii U should only be connected to a WiFi router with no internet access.
10. On the forensic investigator's pc, launch the python RPC program as `python3 -i rpc.py`.
11. On the Wii U, open the web browser and access the payload URL.
12. Invoke `rpc.dump_mem(0x10000000,1000000000,'memorydump.bin')` in the interactive python console. Once the memory dump is completed, properly hash and prepare it for being stored as forensic evidence.

13. Invoke `rpc.dumpfs()`. Hash all the files that have been outputted in the current working directory. Next, prepare those file for being stored as forensic evidence.

14. Invoke `rpc.exit()`.

15. Analyse the Friendlist.

16. Analyse Wii U Chat.

17. Analyse Mii Maker.

18. Analyse the Internet Browser.

19. Repeat steps 15-18 for all accessible users.

20. Connect the Wii U to the internet, but make sure specific URL's that could cause the Wii U to automatically update its firmware are blocked (see [Wii16d] for more information). Perform the next steps for all of the accessible users.

21. Analyse Miiverse.

22. Analyse eShop.

7. Conclusion

In this paper we performed a forensic analysis of the Nintendo Wii U console. The aim of this research was to determine what data can be gathered from the console that could serve as forensic evidence. In order to do so, we focused on two different areas of digital forensics, post mortem and live forensics.

For the post mortem analysis, we studied the architecture of the system and its hardware components. The most relevant components identified were the NAND flash storage and the flash eMMC non volatile storage. However, in order to analyse them properly a chip off was needed. This type of procedure is expensive and requires special tooling. For that reason it was considered to be out of the scope of this project. In addition, we analysed a USB mass storage device containing a Wii U save game. Nonetheless, we were not able to find traces of a file system structure as its contents seem to be either encrypted or highly compressed.

For the live analysis, we showed how to access the Wii U leveraging a user space exploit. This way we were able to access and exfiltrate memory and files. The most relevant finding is a file containing the browser history of a user, as this data is not available through the user interface. We also studied different features of the Wii U such as the Miiverse, and explained what type of forensic data can be extracted from them. Moreover, we found a way of extracting the browser history from one account while being logged in with another account. This implies that even though an account is password protected, it is still possible to access its browser history by either creating a new user or using an existing user without a password.

Finally, we combined this information to create a forensics procedure. This procedure involves commonly known practices such as following the order of volatility, using proper logging and hashing. In addition, the source code used in this research is open source and publicly available, and thus can be verified.

8. Discussion

This research showed how to access data which is not available in the Wii U graphical user interface. However, to access this data an exploit is required. A potential risk is that Nintendo could fix the vulnerability used by this exploit. Still, the findings (like the browser data on disk and traces of URLs in memory) are not likely to change when new firmware versions of the Wii U are released. Also, the homebrew community is very active and new exploits are found soon after the release of a new firmware version. It is important to note that not all exploits are released to the public (some are kept private). As an example, for the latest firmware versions (5.5.0E and 5.5.1E) there is only a user level exploit publicly available [Caf16].

As it can be seen from chapter 5, almost only user data regarding the browsing history was found in memory or on the file system. We are assuming that the Wii U has implemented application separation and a specific application can only access its own data. As the exploit is executed through the web browser, the application separation could explain these findings. Using the `ls` command we were able to distinguish between non-existing and restricted directories, based on the error code returned by the Wii U file operations. Therefore, we could see there were some existing directories which we did not have access to.

We hope that this report can be used as a starting point for developing advanced forensic procedures for the Nintendo Wii U gaming console. This is mainly because our procedure as described in chapter 6, needs to be improved and reviewed by forensic experts before being used in court.

9. Future Work

First of all, more comprehensive testing is required. This involves creating more experiments for different Wii U features. After the execution of those experiments, the next step would be to conduct a thorough analysis of the memory dump and file system structure. Secondly, we would like to enhance the acquisition procedures, by improving the written code for both the memory and the file system dump. Finally, a more detailed description of the forensic procedure involving the graphical user interface features is required.

Bibliography

[BC07] Paul K Burke and Philip Craiger. "Xbox forensics". In: *Journal of Digital Forensic Practice* 1.4 (2007), pp. 275–282.

[Caf16] The Red Hat Cafe. *Wii U System Menu Versions Quick Reference*. Mar. 26, 2016. URL: `http://rhcafe.us.to/`.

[Dav+15] Matthew Davies et al. "Forensic analysis of a Sony PlayStation 4: A first look". In: *Digital Investigation* 12 (2015), S81–S89.

[Fai14] Fail0verflow. *Console Hacking 2013: Omake*. Jan. 2, 2014. URL: `https://fail0verflow.com/blog/2014/console-hacking-2013-omake.html`.

[IGN16] IGN. *Wii U Tech Specs*. Mar. 26, 2016. URL: `http://www.ign.com/wikis/wii-u/Wii_U_Tech_Specs`.

[Jon07] Ryan Jones. "Safer live forensic acquisition". In: *University of Kent at Canterbury* (2007).

[Moo+14] Jason Moore et al. "Preliminary forensic analysis of the Xbox One". In: *Digital Investigation* 11 (2014), S57–S65.

[Nin16a] Nintendo. *System Menu Update History*. Jan. 11, 2016. URL: `http://en-americas-support.nintendo.com/app/answers/detail/a_id/1436/~/system-menu-update-history`.

[Nin16b] Nintendo. *Wii U Internet Browser Specs*. Mar. 26, 2016. URL: `https://www.nintendo.com/wiiu/built-in-software/browser-specs/`.

[NWP16] NWPlayer123. *RPC*. Mar. 26, 2016. URL: `https://gbatemp.net/`.

[Ste10] Peter Stewart. "Forensic Analysis of the Nintendo Wii Game Console". In: (2010).

[Tur08] Benjamin Turnbull. "Forensic investigation of the Nintendo Wii: A first glance". PhD thesis. Citeseer, 2008.

[Wal08] John Walker. *A Pseudorandom Number Sequence Test*. Jan. 28, 2008. URL: `http://www.fourmilab.ch/random/`.

[Wes15] Michael West. "Confronting criminal enterprise in online gaming: Determining awareness and capabilities regarding the emergence of online gaming cybercrime". PhD thesis. UTICA COLLEGE, 2015.

[Wii16a] WiiUBrew. *Cafe OS*. Mar. 26, 2016. URL: http://wiiubrew.org/wiki/Cafe_OS.

[Wii16b] WiiUBrew. *Coreinit.rpl*. Mar. 26, 2016. URL: http://wiiubrew.org/wiki/Coreinit.rpl.

[Wii16c] WiiUBrew. *Nsysnet.rpl*. Mar. 26, 2016. URL: http://wiiubrew.org/wiki/Nsysnet.rpl.

[Wii16d] WiiUBrew. *System Software*. Mar. 26, 2016. URL: http://wiiubrew.org/wiki/System_Software.

[Wii16e] WiiUBrew. *Title database*. Mar. 26, 2016. URL: http://wiiubrew.org/wiki/Title_database.

[Wii16f] Wiiubrew. *Wiiubrew Wiki*. Mar. 28, 2016. URL: http://wiiubrew.org/wiki/Main_Page.

[Wii16g] WiiUDev. *Build system and examples for running C code on the Wii U*. Feb. 28, 2016. URL: https://github.com/wiiudev/libwiiu.

[Wik16a] Wikipedia. *Return-oriented programming*. Mar. 26, 2016. URL: https://en.wikipedia.org/wiki/Return-oriented_programming.

[Wik16b] Wikipedia. *Wii U*. Mar. 26, 2016. URL: https://en.wikipedia.org/wiki/Wii_U.

[Yel16] Yellows8. *ROP-chain-generator for Wii U PowerPC-userland exploits*. Jan. 16, 2016. URL: https://github.com/yellows8/wiiuhaxx_common.